CSCI-GA 2590: Natural Language Processing

Representing and Classifying Text

Name
Parijat Parimal - pp2206

Collaborators: None
By turning in this assignment, I agree by the honor code of the College of Arts and Science at New York University and declare that all of this is my own work.

Welcome to your first assignment! The goal of this assignment is to get familiar with basic text classification models through both mathematical analysis and hands-on feature engineering. **Before you get started, please read the Submission section thoroughly**.

Submission

Submission is done on Gradescope.

Written: When submitting the written parts, make sure to select **all** the pages that contain part of your answer for that problem, or else you will not get credit. You can either directly type your solution between the `shaded` environments in the released `.tex` file, or write your solution using pen or stylus. A `.pdf` file must be submitted.
Programming: Questions marked with "coding" next to the assigned to the points require a coding part in `submission.py`. Submit `submission.py` and we will run an autograder on Gradescope. You can use functions in `util.py`. However, please do not import additional libraries (e.g. `numpy`, `sklearn`) that aren't mentioned in the assignment, otherwise the grader may crash and no credit will be given. You can run `test.py` to test your code but you don't need to submit it. Please do not change the name of the functions as it might cause the grader to crash. Also after running the autograder, the graders will manually check the code. No credits will be awarded for hard coding the answers

Problem 1: Naive Bayes classifier

In this problem, we will study the *decision boundary* of multinomial Naive Bayes model for binary text classification. The decision boundary is often specificed as the level set of a function: $\{x \in \mathcal{X} : h(x) = 0\}$, where x for which $h(x) > 0$ is in the positive class and x for which $h(x) < 0$ is in the negative class.

1. [2 points] Give an expression of $h(x)$ for the Naive Bayes model $p_\theta(y \mid x)$, where θ denotes parameters of the model.

   Ans:

   $$h(x) = log\frac{p_\theta(y = + \mid x)}{p_\theta(y = - \mid x)}$$

   If probability of y = + outweighs probability for y = - for input x, log value will be positive, if both are equal, it will be 0 and negative otherwise. Thus, h(x) = 0 will give the decision boundary and $h(x) > 0$ will indicate positive class and $h(x) < 0$ will indicate negative class.

2. [3 points] Recall that for multinomial Naive Bayes, we have the input $X = (X_1, \ldots, X_n)$ where n is the number of words in an example. In general, n changes with each example but we can ignore that for now. We assume that $X_i \mid Y = y \sim \text{Categorical}(\theta_{w_1,y}, \ldots \theta_{w_m,y})$ where $Y \in \{0, 1\}$, $w_i \in \mathcal{V}$, and $m = |\mathcal{V}|$ is the vocabulary size. Further, $Y \sim \text{Bernoulli}(\theta_1)$. Show that the multinomial Naive Bayes model has a linear decision boundary, i.e. show that $h(x)$ can be written in the form $w \cdot x + b = 0$. [**RECALL:** The categorical distribution is a multinomial distribution with one trial. Its PMF is

$$p(x_1, \ldots, x_m) = \prod_{i=1}^{m} \theta_i^{x_i} \ ,$$

where $x_i = \mathbb{1}[x = i]$, $\sum_{i=1}^{m} x_i = 1$, and $\sum_{i=1}^{m} \theta_i = 1$.]

Ans:

$$h(x) = log\frac{p(y = + \mid x)}{p(y = - \mid x)}$$

$$log\frac{p(y = + \mid x)}{p(y = - \mid x)} = log(p(y = + \mid x)) - log(p(y = - \mid x)) \qquad - Eq(1)$$

Expanding for y = +

$$log(p(y = + \mid x)) = log(p(x \mid y = +).p(y = +)) = log(p(x \mid y = +)) + log(p(y = +))$$

Expanding multinomial probablity

$$= log(\prod_{i=1}^{m} \theta_{i+}^{x_i}) + log(p(y = +)) = \sum_{i=1}^{m} (x_i log(\theta_{i+})) + log(p(y = +))$$

Similarly, for y = -

$$log(p(y = - \mid x)) = \sum_{i=1}^{m} (x_i log(\theta_{i-})) + log(p(y = -))$$

Using these in Eq(1)

$$h(x) = \sum_{i=1}^{m} (x_i log(\theta_{i+})) + log(p(y = +)) - \sum_{i=1}^{m} (x_i log(\theta_{i-})) + log(p(y = -))$$

$$= \sum_{i=1}^{m} x_i [log(\theta_{i+}) - log(\theta_{i-})] + log(p(y = +)) - log(p(y = -))$$

Using x as vector to represent x_i for all $i \epsilon [1, m]$ and w to represent vector $[log(\theta_{i+}) - log(\theta_{i-})]$ for all $i \epsilon [1, m]$, such that the dot product of w and x gives $\sum_{i=1}^{m} x_i [log(\theta_{i+}) - log(\theta_{i-})]$. Also, let b = $log(p(y = +)) - log(p(y = -))$. since p(y=+) and p(y=-) do not depend on x, we van treat them as constant. For the decision boundary, h(x) = 0 and using the above expression with the vectors w, x and constant b specified above, we get,

$$h(x) = w.x + b = 0$$

Hence the decision boundary is linear.

3. [2 points] In the above model, X_i represents a single word, i.e. it's a unigram model. Think of an example in text classification where the Naive Bayes assumption might be violated. How would you allieviate the problem?

   Ans: Our Naive Bayes assumption is to consider X_i as conditionally independent of X_j where $i \neq j$. For an input sentence "This was not good" vs "This was very good", even though the meaning for word "good" as a word is same, both the sentences have opposite meaning. But with our unigram model, we do not consider the impact of "not" or "very" on the word "good", which alters the meaning of the sentence completely. Thus instead of unigram model, we could use a bigram model where we consider a pair of words to make a feature instead of a single word. That would make "not good" and "very good" as features and the meaning derived thereof will alleviate the problem we had with unigram model.

4. [2 points] Since the decision boundary is linear, the Naive Bayes model works well if the data is linearly separable. Discuss ways to make text data more separable and its influence on model generalization.

   As in the example mentioned above, we could see that a bigram model might work better in above example. We can also come up with similar situations where we may more words are interdependent on each other. There could be examples where where we might need multigram model, but we will have to consider the trade-off between computational expense vs accuracy of our model. And then we will have to find a suitable range of words to consider their interdependence. This can be done by considering 'k' neighbouring words to make features, where 'k' could be chosen based on expectations of interdependence of words.

   Also, we can make a separate dictionary which might contain insignificant information and ignore them while extracting features. For example, all documents may have "the", "is", "are" etc which may not be useful to label the documents, thus, ignoring these words will make better distinction of labels with respect to the features. Thus making the model more separable.

Problem 2: Skip-gram model

In this problem, we will gain some insights into the skip-gram model. In particular, we will show that it encourages the inner product between two word embeddings to be equal to their pointwise mutual information (PMI) (up to a constant).
Recall that to evaluate the objective function of the skip-gram model we need to enumerate over the entire vocabulary $\mathcal{V}$, which can be quite expensive in practice. Note that the reason we need to enumrate the vocabulary is to obtain a probability distribution of neighboring words. Let's define the *neighborhood* of a word w to be a size-L window around it and words within the window are w's neighbors. Instead of modeling the distribution of words, we can model the distribution of an indicator: whether two words are neighbors. Let Y be the indicator random variable. We model it by a Bernoulli distribution

$$p_\theta(y = 1 \mid w, c) = \frac{1}{1 + e^{-u_c \cdot v_w}} ,$$

where $c \in \mathcal{V}$ is within a window centered at $w \in \mathcal{V}$, u_c, v_w represent embeddings of c and w, and θ is the model parameter, i.e. word embedding v's and context embedding u's. What about the negative observations ($y = 0$)? A naive way is to consider all words that do not occur in the neighborhood of w, however, this is as expensive as our original objective. One solution is **negative sampling**, where we only consider a small number of non-neighboring words.
Let $D = \{(w, c)\}$ be the set of all word-neighbor pairs observed in the training set. For each pair (w, c), we generate k negative neighbors c_N by sampling from a distribution $p_N(\cdot)$ over the vocabulary.

1. [3 points] Write down a learning objective that maximizes the log-likelihood of D and the *expected* log-likelihood of the negative examples. Note that C_N is the random variable in the expectation. You can use the Bernoulli model $p_\theta(y \mid w, c)$ in the expression.

   Ans:
   Log-likelihood of D is

   $$log \prod_{(w,c)\in D} p_\theta(y = 1 \mid w, c) = \sum_{(w,c)\in D} log \frac{1}{1 + e^{-u_c \cdot v_w}}$$

   Expected log-likelihood of negative examples for each $(w, c) \in D$ is

   $$\log \prod_k E[C_N] = \sum_k \log p_N(c_k)$$

   Since probability of any c to be a negative example = 1 - $p_\theta(y = 1 \mid w, c)$,

   $$p_N(c) = 1 - p_\theta(y = 1 \mid w, c) = 1 - \frac{1}{1 + e^{-u_c \cdot v_w}} = \frac{1}{1 + e^{u_c \cdot v_w}}$$

   Using this in equation for expected log-likelihood of negative examples for $each(w, c)$, we get

   $$\sum_k log \frac{1}{1 + e^{u_{c_k} \cdot v_w}}$$

   Adding log-likelihood for positive and k negative examples and maximizing it gives our learning objective:

   $$max[\sum_{(w,c)\in D} (log \frac{1}{1 + e^{-u_c \cdot v_w}} + \sum_k log \frac{1}{1 + e^{u_{c_k} \cdot v_w}})]$$

2. [2 points] Let

$$\sigma(\alpha) = \frac{1}{1+e^{-\alpha}} \ .$$

Show that

$$\frac{d}{d\alpha} \log \sigma(\alpha) = \frac{1}{1+e^{\alpha}} \ .$$

Ans:

$$\frac{d}{d\alpha} \log \sigma(\alpha) = \frac{d}{d\alpha} log(\frac{1}{1+e^{-\alpha}})$$

$$= \frac{d}{d\alpha}(-log(1+e^{-\alpha}))$$

By chain rule,

$$= \frac{-1}{1+e^{-\alpha}} \frac{d}{d\alpha}(1+e^{-\alpha})$$

$$= \frac{-1}{1+e^{-\alpha}}(0+(-e^{-\alpha}))$$

Multiplying and dividing by e^{α},

$$= \frac{e^{-\alpha}}{1+e^{-\alpha}} = \frac{e^{-\alpha}}{1+e^{-\alpha}} \cdot \frac{e^{\alpha}}{e^{\alpha}}$$

$$= \frac{1}{e^{\alpha}+1} = \frac{1}{1+e^{\alpha}}$$

Hence proved,

$$\frac{d}{d\alpha} \log \sigma(\alpha) = \frac{1}{1+e^{\alpha}}$$

3. [3 points] Let's consider one pair $(w = b, c = a)$. Let $\alpha = u_a \cdot v_b$. Show that the optimal solution of α is

$$\alpha^* = \log \frac{A}{B} ,$$

where

$$A = \sum_{(w,c)\in D} \mathbb{1}[w = b, c = a] \tag{1}$$

$$B = k \sum_{(b,\cdot)\in D} p_N(a) . \tag{2}$$

[**HINT:** Let $\ell(\theta)$ be the objective. Solve for α in $\frac{\partial \ell}{\partial \alpha} = 0$. You can use results from previous questions.]

Ans:
For $\alpha = u_a \cdot v_b$, our learning objective $\ell(\theta)$ is:

$$max[\sum_{(w,c)\in D} (log \frac{1}{1 + e^{-u_c \cdot v_w}} + \sum_k log \frac{1}{1 + e^{u_{c_k} \cdot v_w}})]$$

Since $\frac{\partial \ell}{\partial \alpha} = 0$ gives optimum value for α. Using equations obtained from part 2, we obtain the following derivative:

$$\frac{\partial \ell}{\partial \alpha} = \sum_{(w,c)\in D} \frac{1}{1 + e^{\alpha}} + k \sum_{(w,.)\in D} \frac{1}{1 + e^{-u_{c_k} \cdot v_w}} = 0 \qquad - EQ(3)$$

For $(w = b, c = a)$,

$$k \sum_{(w,.)\in D} \frac{1}{1 + e^{-u_{c_k} \cdot v_w}} = Probability\ of\ k\ negative\ words\ for\ all\ (b, a) \in D$$

$$= \frac{count\ of\ k\ negative\ examples\ for\ a}{count\ of\ all\ pairs\ (b, a) \in D} = \frac{k \sum_{(b,\cdot)\in D} p_N(a)}{\sum_{(w,c)\in D} \mathbb{1}[w = b, c = a]}$$

Using above to solve EQ(3), we get

$$\frac{1}{e^{\alpha}} = \frac{k \sum_{(b,\cdot)\in D} p_N(a)}{\sum_{(w,c)\in D} \mathbb{1}[w = b, c = a]} = \frac{B}{A}$$

Thus,

$$e^{\alpha} = \frac{A}{B}$$

Hence, the optimal value for $\alpha = \alpha*$ is:

$$\alpha* = \log \frac{A}{B}$$

4. [2 points] Suppose the distribution of negative words $p_N(w)$ for $w \in \mathcal{V}$ is a categorical distribution given by

$$p_N(w) = \frac{\text{count}(w, \cdot, D)}{|D|} \ ,$$

where $\text{count}(w, \cdot, D) = \sum_{c \in \mathcal{V}} \text{count}(w, c, D)$. Note that $p_N(w)$ is simply the fraction of unigram w in D. Show that

$$\alpha^* = \text{PMI}(b, a) - \log k \ ,$$

where the PMI score of word co-occurence in D is defined as

$$\text{PMI}(b, a) \stackrel{\text{def}}{=} \log \frac{\text{count}(b, a, D)|D|}{\text{count}(b, \cdot, D)\text{count}(a, \cdot, D)} \ .$$

Ans:
From Part 3, we have $\alpha* = \log \frac{A}{B}$,
Expanding A, we get,

$$\sum_{(w,c) \in D} \mathbb{1}[w = b, c = a] = P_\theta(b, a|D) = \frac{count(b, a, d)}{|D|} \qquad - Eq(1)$$

Expanding B, for k negative examples for 'a' and 'b', we get,

$$k \sum_{(b,\cdot) \in D} p_N(a) = k.(P_\theta(b, .|D).P_\theta(a, .|D)) = k.\frac{count(b, ., D)}{|D|}.\frac{count(a, ., D)}{|D|} \qquad - EQ(2)$$

Using Eq(1) and Eq(2), we get,

$$\frac{A}{B} = \frac{\frac{count(b,a,d)}{|D|}}{k.\frac{count(b,.,D)}{|D|}.\frac{count(a,.,D)}{|D|}} = \frac{\text{count}(b, a, D)|D|}{\text{k}.count(b, \cdot, D)\text{count}(a, \cdot, D)}$$

Taking log both sides,

$$\log \frac{A}{B} = log \frac{\text{count}(b, a, D)|D|}{\text{k}.count(b, \cdot, D)\text{count}(a, \cdot, D)}$$

$$= log \frac{\text{count}(b, a, D)|D|}{\text{count}(b, \cdot, D)\text{count}(a, \cdot, D)} - \log k$$

Since, $\text{PMI}(b, a) \stackrel{\text{def}}{=} \log \frac{\text{count}(b,a,D)|D|}{\text{count}(b,\cdot,D)\text{count}(a,\cdot,D)}$, we have,

$$\alpha* = \log \frac{A}{B} = log \frac{\text{count}(b, a, D)|D|}{\text{count}(b, \cdot, D)\text{count}(a, \cdot, D)} - \log k = \text{PMI}(b, a) - \log k$$

Hence,

$$\alpha^* = \text{PMI}(b, a) - \log k$$

5. [2 points] Let's denote the unigram model defined above by $p_{\text{unigram}}(w)$. In practice, we often prefer to use $p_N(w) \sim p^{\beta}_{\text{unigram}}(w)$ where $\beta \in [0, +\infty]$. Describe the desired range of β and explain why.

   Ans:
   The desired value for β should be to add weight to uncommon words, such that usage of such word will give a more clear idea of the context (Example - gene, DNA) in which it was used. Also, it should not increment the weight of words that are too frequent, as there is not much impact of those words in deciphering the context (Example - is, the). To do that, β should be a fraction and thus the desired range of β should be [0,1]. If β is greater than 1, the weights on uncommon words will be diminished. Thus, β should not be greater than 1.

   Hence, desired $\beta \in [0, 1]$

Problem 3: Natural language inference

In this problem, you will build a logistic regression model for textual entailment. Given a *premise* sentence, and a *hypothesis* sentence, we would like to predict whether the hypothesis is *entailed* by the premise, i.e. if the premise is true, then the hypothesis must be true.
Example:

label	premise	hypothesis
entailment	The kids are playing in the park	The kids are playing
non-entailment	The kids are playing in the park	The kids are happy

1. [1 point] Given a dataset $D = \{(x^{(i)}, y^{(i)})\}_{i=1}^{n}$ where $y \in \{0, 1\}$, let ϕ be the feature extractor and w be the weight vector. Write the maximum log-likelihood objective as a *minimization* problem.

   Ans:
   Negative log likelihood can be expressed as:

$$l(w) = -\frac{1}{n}\sum_{i=1}^{n}[y^{(i)}.\log\frac{1}{1+e^{-w\cdot\phi(x^{(i)})}} + (1-y^{(i)}).\log(1-\frac{1}{1+e^{-w\cdot\phi(x^{(i)})}})]$$

   Maximum log-likelihood as minimization problem is to minimize the negative log-likelihood:

$$min\ l(w) = min[-\frac{1}{n}\sum_{i=1}^{n}[y^{(i)}.\log\frac{1}{1+e^{-w\cdot\phi(x^{(i)})}} + (1-y^{(i)}).\log(1-\frac{1}{1+e^{-w\cdot\phi(x^{(i)})}})]]$$

2. [3 point, coding] We first need to decide the features to represent x. Implement `extract_unigram_features` which returns a BoW feature vector for the premise and the hypothesis.

3. [2 point] Let $\ell(w)$ be the objective you obtained above. Compute the gradient of $\ell_i(w)$ given a single example $(x^{(i)}, y^{(i)})$. Note that $\ell(w) = \sum_{i=1}^{n} \ell_i(w)$. You can use $f_w(x) = \frac{1}{1+e^{-w\cdot\phi(x)}}$ to simplify the expression.

Ans:
Gradient of $l_i(w)$ is given as:

$$\frac{\partial l_i(w)}{\partial w} = -\frac{\partial}{\partial w}[y^{(i)}.\log\frac{1}{1+e^{-w\cdot\phi(x^{(i)})}} + (1-y^{(i)}).\log(1-\frac{1}{1+e^{-w\cdot\phi(x^{(i)})}})]$$

$$= -\frac{\partial}{\partial w}[y^{(i)}.\log f_w(x^{(i)}) + (1-y^{(i)}).\log(1-f_w(x^{(i)}))]$$

$$= -[y^{(i)}.\frac{\partial}{\partial w}(\log f_w(x^{(i)})) + (1-y^{(i)}).\frac{\partial}{\partial w}(\log(1-f_w(x^{(i)})))] \qquad -Eq(1)$$

Now we compute the partial derivatives separately (using results from Question 1 and 2 of Part 2), we get,

$$\frac{\partial}{\partial w}(\log f_w(x^{(i)})) = \frac{\phi(x^{(i)}).e^{-w\cdot\phi(x^{(i)})}}{1+e^{-w\cdot\phi(x^{(i)})}} = \phi(x^{(i)}).(1-f_w(x^{(i)}))$$

Similarly, we get,

$$\frac{\partial}{\partial w}(1-\log f_w(x^{(i)})) = -\phi(x^{(i)}).f_w(x^{(i)})$$

Combining above derivatives in Eq(1), we get,

$$\frac{\partial l_i(w)}{\partial w} = -[y^{(i)}.\phi(x^{(i)}).(1-f_w(x^{(i)})) + (1-y^{(i)}).(-\phi(x^{(i)}).f_w(x^{(i)}))]$$

Simplifying the above equation, we get,

$$\frac{\partial l_i(w)}{\partial w} = -\phi(x^{(i)}).(y^{(i)} - f_w(x^{(i)}))$$

Hence the gradient for single example $(x^{(i)}, y^{(i)})$ is:

$$-\phi(x^{(i)}).(y^{(i)} - f_w(x^{(i)})) \qquad [where,\ f_w(x^{(i)}) = \frac{1}{1+e^{-w\cdot\phi(x^{(i)})}}]$$

4. [5 points, coding] Use the gradient you derived above to implement `learn_predictor`. You must obtain an error rate less than 0.3 on the training set and less than 0.4 on the test set to get full credit *using the unigram feature extractor*.

5. [3 points] Discuss what are the potential problems with the unigram feature extractor and describe your design of a better feature extractor.

   Ans:
   Unigram feature extractor gives equal value to each word, but there are many situations where some words might have more value in choosing a label. In the specific Premise and hypothesis problem, it is can be seen that for each example pair, Premise is substantially larger than Hypothesis. To quantify that, overall number of words in premise is nearly twice that of Hypothesis. For our prediction to be more accurate, we want both Premise and Hypothesis to have equal values, but since the word count in Premise is twice that of Hypothesis, features extracted from Premise outweigh features extracted from Hypothesis.

   To overcome this bias of Premise over Hypothesis, we count each word in Hypothesis twice to compensate for this bias. Thus, our feature extractor will have a bias for Hypothesis to compensate for the inherent bias in input data for Premise. Hence, this biased scaling of features will lead to equal valued features, consequently enhancing the overall accuracy of the algorithm.

6. [3 points, coding] Implement your feature extractor in `extract_custom_features`. You must get a lower error rate on the dev set than what you got using the unigram feature extractor to get full credits.

7. [3 points] When you run the tests in `test.py`, it will output a file `error_analysis.txt`. (You may want to take a look at the `error_analysis` function in `util.py`). Select five misclassified examples. For each example, briefly state your intuition for why the classifier got it wrong, and what information about the example will be needed to get it right.

   Ans:
   Wrongly classified examples:

   1. P: An older gentleman speaking at a podium . H: A man giving a speech
   label=0, predicted=1, wrong
   The algorithm was not able to understand that "gentleman" and "man" refer to same thing. Same with "speaking" and "speech". To resolve this, we will need information about word roots and map words with same roots as same words.

   2. P: A building that portrays beautiful architecture stands in the sunlight as somebody on a bike passes by . H: A bicyclist passes an esthetically beautiful building on a sunny day
   label=1, predicted=0, wrong
   Algorithm could not understand ("Bike","Bicyclist") and ("sunlight","sunny") to have same reference. Here as well, we will need to have roots of words to extract features.

   3. P: A woman with red-hair and wearing a black tank top and short jeans shorts appears to be yelling . H: A woman with red-hair and wearing a black tank top screams at her son who is crossing the road without looking both ways . label=0, predicted=1, wrong
   From the data provided, even for human reader, the two sentences seem to have related context and the algorithm does a fine job in predicting label 1. The information needed to correctly label this example is to have more data in the example itself that would help to distinguish the context of the two sentences.

   4. P: Two guys are wearing uniforms who are running in the grass . H: Two guys in uniforms are playing a sport on the grass label=0, predicted=1, wrong
   Since single words "running" and "playing" change the context, we need to have a list of words that have more weight. With this information, we could add a bias for these words (Example: each word that represents a verb can be counted twice or thrice to add the bias).

   5. P: A man wearing an apron is smiling at a plate of food . H: The man is happy by the sight of the food before him .
   label=1, predicted=0, wrong
   Here "happy by the sight" and "smiling" have the same context. We will need a map of phrases or group of words to a sentiment. If the sentiment matches, that could be considered as a match and that needs to be incorporated as a feature to rectify errors like this.

8. [3 points] Change `extract_unigram_features` such that it only extracts features from the hypothesis. How does it affect the accuracy? Is this what you expected? If yes, explain why. If not, give some hypothesis for why this is happening. Don't forget to change the function back before your submit. You do not need to submit your code for this part.

   Ans:
   If only Hypothesis is used to extract features, the accuracy for training data is almost same (0.251 to 0.2468), but the accuracy on validation data improved significantly (0.36 to 0.28). When we look at the data sets, it can be seen that Hypothesis in general are smaller sentences containing more significant words, while Premise contains a lot of Stop words.

   Extracting features only from Hypothesis has two major consequences. First is that we get more refined features as we have less stop words, hence the features are scaled better (as more significant words get more weights than stop words).

   Secondly, when all of training data is to extract features, the noise is also considered as features to find weights/parameters. This can lead to overfitting as the same noise may not be present in test or validation data. Hence, with lesser data from training set, yet keeping the more significant data to extract features, reduces overfitting as can be seen by the significant improvement of accuracy in validation data.